Package 'urbin'

July 7, 2025

Version 0.1-16

Date 2025-07-06

Title Unifying Estimation Results with Binary Dependent Variables

Depends R (>= 2.14.0)

Suggests sampleSelection (>= 0.7-0), maxLik (>= 1.1-2), mfx (>= 1.1), mlogit (>= 0.3-0), MASS (>= 7.3-50), mvProbit (>= 0.1-8), knitr, stargazer

VignetteBuilder knitr

Description Calculate unified measures
that quantify the effect of a covariate on a binary dependent variable
(e.g., for meta-analyses).
This can be particularly important
if the estimation results are obtained
with different models/estimators
(e.g., linear probability model, logit, probit, ...)
and/or with different transformations of the explanatory variable of interest
(e.g., linear, quadratic, interval-coded, ...).
The calculated unified measures are:
(a) semi-elasticities of linear, quadratic, or interval-coded covariates and
(b) effects of linear, quadratic, interval-coded, or categorical covariates
when a linear or quadratic covariate changes between distinct intervals,
the reference category of a categorical variable
or the reference interval of an interval-coded variable needs to be changed,
or some categories of a categorical covariate
or some intervals of an interval-coded covariate need to be grouped together.
Approximate standard errors of the unified measures are also calculated.
All methods that are implemented in this package
are described in the 'vignette'
``Extracting and Unifying Semi-Elasticities and Effect Sizes
from Studies with Binary Dependent Variables"
that is included in this package.

License GPL (>= 2)

URL https://github.com/arne-henningsen/urbin/

NeedsCompilation no

Author Arne Henningsen [aut, cre] (ORCID:
<https://orcid.org/0000-0002-6720-0264>),
Geraldine Henningsen [aut] (ORCID:
<https://orcid.org/0000-0002-4651-1039>)

Maintainer Arne Henningsen <arne.henningsen@gmail.com>

Repository CRAN

Date/Publication 2025-07-06 22:30:02 UTC

Contents

`urbin` *Unifying Estimation Results with Binary Dependent Variables*

Description

These four functions calculate the semi-elasticities and effects of explanatory variables in linear probability models, binary probit models, ordered probit models, multivariate probit models, binary logit models, and multinomial logit models.

`urbinEla()` calculates the semi-elasticity of a continuous variable that is used as a linear explanatory variable or as a linear and quadratic explanatory variable.

`urbinElaInt()` calculates the semi-elasticity of an interval-coded explanatory variable.

`urbinEffInt()` calculates the effect of a continuous variable that is used as a linear explanatory variable or as a linear and quadratic explanatory variable if this variable changes between discrete intervals.

`urbinEffCat()` calculates the effect of a categorical variable that is used as an explanatory variable, particularly if one wants to change the reference category and/or wants to group some of the categories together to a new category.

The semi-elasticities calculated by `urbinEla()` and `urbinElaInt()` indicate by how many percentage points the probability that the dependent variable has a value of one increases if the explanatory variable of interest increases by one percent.

The effects calculated by `urbinEffInt()` and `urbinEffCat()` indicate by how much the probability that the dependent variable has a value of one increases if the explanatory variable of interest changes from the 'reference' interval/category to a selected interval/category of interest (this effect multiplied by 100 indicates the increase in percentage points).

The four functions apply the Delta-method to calculate the approximate standard errors of the calculated semi-elasticities and effects.

Usage

```
urbinEla( allCoef, allXVal, xPos, model, allCoefVcov = NULL,
  seSimplify = !is.matrix( allCoefVcov ), xMeanSd = NULL,
  iPos = 1, yCat = NULL )

urbinElaInt( allCoef, allXVal, xPos, xBound, model,
  allCoefVcov = NULL, iPos = 1, yCat = NULL )

urbinEffInt( allCoef, allXVal = NULL, xPos, refBound, intBound, model,
  allCoefVcov = NULL, xMeanSd = NULL, iPos = 1, yCat = NULL )

urbinEffCat( allCoef, allXVal, xPos, xGroups, model,
  allCoefVcov = NULL, iPos = 1, yCat = NULL )
```

Arguments

`allCoef`	a vector of all estimated coefficients (including intercept).
`allXVal`	a vector of the values of the explanatory variables, at which the semi-elasticity or the effect should be calculated (including intercept; the order of its elements must be the same as the order of the corresponding coefficients in argument `allCoef`).
`xPos`	a vector of non-negative integers indicating the position(s) of the coefficient(s) and value(s) of the explanatory variable of interest in arguments `allCoef`, `allXVal`, and eventually `allCoefVcov` (for `urbinElaInt()` and `urbinEffCat()` see **Details** section below).
`xBound`	a numeric vector indicating the boundaries of the intervals.
`refBound`	a numeric vector of two elements that indicate the lower boundary and the upper boundary of the 'reference' interval, respectively.
`intBound`	a numeric vector of two elements that indicate the lower boundary and the upper boundary of the interval of interest, respectively.
`xGroups`	a vector consisting of -1s, 0s, and 1s, where a -1 indicates that the category should belong to the new reference category, a 1 indicates that the category should belong to the new category of interest, and a zero indicates that the category belongs to neither; the length of this vector must be equal to the number of categories; the last element of this vector corresponds to the reference category, while all other elements correspond to the categories as indicated by argument `xPos`.
`model`	a character string that indicates the type of model that was estimated to obtain the parameter estimates: `"lpm"` = linear probability model (see **Details** section below), `"probit"` = binary probit model or multivariate probit model (for multivariate probit models see **Details** section below), `"oprobit"` = ordered probit model (see **Details** section below), `"logit"` = binary logit model, or `"mlogit"` = multinomial logit model (see **Details** section below).
`allCoefVcov`	an optional argument that can be the variance-covariance matrix of all estimated coefficients OR the vector of the standard errors of all estimated coefficients (including intercept; the order of these values must be the same as the order of the corresponding coefficients in argument `allCoef`).

`seSimplify` logical value indicating whether the standard errors of the semi-elasticities should be calculated by a simplified equation (i.e., assuming that $\phi(\beta' x)$ does not depend on β), which may be more appropriate if the off-diagonal elements of the variance-covariance matrix are unknown.

`xMeanSd` a vector with two elements: the mean value and the standard deviation of the explanatory variable of interest. If argument `allCoefVcov` is a vector of standard errors and the explanatory variables include a linear and a quadratic term of the variable of interest, the information provided in argument `xMeanSd` is used to approximate the covariance between the coefficient of the linear term and the coefficient of the quadratic term.

`iPos` a posivite integer indicating the position of the intercept in arguments `allCoef`, `allXVal`, and eventually `allCoefVcov`. A value of zero indicates that the model does not have an intercept.

`yCat` a non-negative integer or a vector of non-negative integers that indicate(s) which category/categories of the dependent variable of a multinomial logit model should indicate a binary outcome of one. A zero indicates the base category of the dependent variable (see **Details** section below).

Details

Argument `xPos` **of** `urbinElaInt()` must be a vector of non-negative integers with length equal to the number of intervals. Each element of this vector must refer to one interval, whereas these intervals must be in ascending order. Each element of this vector indicates the position of the coefficient of the respective dummy variable (i.e., the dummy variable that indicates whether the values of the explanatory variable of interest lie in the corresponding interval or not) in arguments `allCoef` and the position of the value of this dummy variable (i.e., share of observations that lie in the respective interval in the sample of interest) in argument `allXVal`. The position of the reference interval (i.e., the interval without a corresponding dummy variable as explanatory variable) must be indicated by a zero.

Argument `xPos` **of** `urbinEffCat()` must be a vector of posivite integers that indicates the positions of the coefficients of the categorical variable of interest in argument `allCoef` and, equally, the positions of the shares of the non-reference categories in argument `allXVal`. This vector must have one element less than the number of categories, because argument `allCoef` does not include a coefficient of the reference category.

`urbinEffInt()` **ignores argument** `allXVal` in case of linear probability models. In case of all other types of models, `urbinEffInt()` ignores the element(s) of argument `allXVal` that correspond to the variable of interest; these values can be set to `NA`.

Linear Probability Model: If a user wants to calculate the semi-elasticity or effect of a variable of interest based on a linear probability model using `urbin*( ..., model = "lpm")`, he/she can include only the coefficient(s) of the explanatory variable of interest in argument `allCoef` and omit all other coefficients. In this case, arguments `allXVal` and `allCoefVcov` must be adjusted accordingly. When using `urbinEla()`, argument `allXVal` can be a scalar with the value of the variable of interest even if the model includes both a linear and a quadratic term. This simplified use of the `urbin*()` functions is only unsuitable for linear probability models and unsuitable for all other types of models.

Ordered Probit Model: The dependent variable has $P > 2$ distinct and strictly ordered categories. In order to aggregate these P cateories into a binary outcome, a user needs to choose a number

$p \in \{1, \ldots, P-1\}$ that separates the P ordered categories into two mutually exclusive combined categories, where the lower p categories indicate a binary outcome of zero and the upper $P - p$ categories indicate a binary outcome of one. An ordered probit model does not have an intercept but it has $P - 1$ thresholds between the P ordered categories. If a user wants to calculate the semi-elasticity based on an ordered probit model using `urbinEla( ..., model = "oprobit")`, argument `allCoef` must include both the coefficients and the thresholds. Consequently, argument `allCoefVcov` must also include the variences and covariances or the standard arrors of both the coefficients and the thresholds. The vector specified by argument `allXVal` must include elements that correspond to the thresholds, where the element that corresponds to the p's threshold must be equal to -1, while the elements that correspond to the other thresholds must be equal to 0. Argument `iPos` must specify the position of the p's threshold in arguments `allCoef`, `allXVal`, and eventually `allCoefVcov`. While the p's threshold must be specified in argument `allCoef`, all other thresholds can be omitted. In this case, arguments `allXVal` and `allCoefVcov` must be adjusted accordingly.

Multivariate Probit Model: This type of model has $P \geq 2$ dependent variables. The `urbin*()` functions can calculate unconditional (but not conditional) semi-elasticities and effects based on the estimation results from multivariate probit models. Argument `allCoef` must include all coefficients of the regression equation of the dependent variable of interest, while the coefficients of the regression equations of the other dependent variables as well as the correlation parameters must be omitted. Argument `allCoefVcov` must be specified accordingly.

Multinomial Logit Model: The dependent variable has $P > 2$ distinct categories (without any logical order). For each category of the dependent variable (except for the base category), the multinomial logit model has a different set of coefficients. Argument `allCoef` must be a vector of all these coefficients with the following ordering: `c(` coefficients of all explanatory variables for the first category of the dependent variable, coefficients of all explanatory variables for the second category of the dependent variable, ..., coefficients of all explanatory variables for the last category of the dependent variable `)`, where the coefficients of the base category (that are all normalized to zero) should not be included in argument `allCoef`. The order of the coefficients for each and every category must be the same as the order of the corresponding explanatory variables in argument `allXVal`. The order of the coefficients in argument `allCoefCov` must be the same as the order of the coefficients in argument `allCoef`. In order to aggregate the P cateories of the dependent variable into a binary outcome, a user needs to use argument `yCat` to select $p \in \{1, \ldots, P-1\}$ categories that should indicate a binary outcome of one, where a zero indicates the base category. All other categories, i.e., all categories that are not specified by argument `yCat`, indicate a binary outcome of zero.

Marginal Effects: If you know the marginal effects of the explanatory variable(s) of interest on the dependent variable, you can use these marginal effects as if they were coefficients of a linear probability model and calculate approximate semi-elasticities and effects with `urbin*( ..., model = "lpm" )` with argument `allCoef` equal to the marginal effect(s) and eventually argument `allCoefVcov` equal to the variance covariance matrix or the standard errors of the marginal effects.

Value

The four `urbin*()` functions return a list of class `"urbin"` that contains the following components:

`call`	the matched call.
`semEla`	the calculated semi-elasticity (only `urbinEla()` and `urbinElaInt()`).
`effect`	the calculated effect (only `urbinEffInt()` and `urbinEffCat()`).

stdEr	the approximate standard error of the calculated semi-elasticity or effect.
allCoefVcov	the variance covariance matrix that was used to calculate the approximate standard error of the calculated semi-elasticity or effect.
derivCoef	the partial derivatives of the semi-elasticity or effect with respect to the coefficients that was used to calculate the approximate standard error of the calculated semi-elasticity or effect, respectively.

Examples

```
# load data set
data( "Mroz87", package = "sampleSelection" )

# create dummy variable for kids
Mroz87$kids <- as.numeric( Mroz87$kids5 > 0 | Mroz87$kids618 > 0 )

# estimate probit model with linear and quadratic age terms
estProbitQuad <- glm( lfp ~ kids + age + I(age^2) + educ,
  family = binomial(link = "probit"),
  data = Mroz87 )
summary( estProbitQuad )

# mean values of the explanatory variables
xMeanQuad <- c( 1, mean( Mroz87$kids ), mean( Mroz87$age ),
  mean( Mroz87$age )^2, mean( Mroz87$educ ) )

# create dummy variables for age intervals
Mroz87$age30.37 <- Mroz87$age >= 30 & Mroz87$age <= 37
Mroz87$age38.44 <- Mroz87$age >= 38 & Mroz87$age <= 44
Mroz87$age45.52 <- Mroz87$age >= 45 & Mroz87$age <= 52
Mroz87$age53.60 <- Mroz87$age >= 53 & Mroz87$age <= 60

# probit estimation with age as interval-coded explanatory variable
estProbitInt <- glm( lfp ~ kids + age30.37 + age38.44 + age53.60 + educ,
  family = binomial(link = "probit"),
  data = Mroz87 )
summary( estProbitInt )

# mean values of the explanatory variables
xMeanInt <- c( 1, colMeans( Mroz87[ ,
  c( "kids", "age30.37", "age38.44", "age53.60", "educ" ) ] ) )


######################################################################
#####################  urbinEla()  ###################################
######################################################################
# semi-elasticity of age (full covariance matrix of coefficients)
urbinEla( coef( estProbitQuad ), xMeanQuad, c( 3, 4 ), model = "probit",
  vcov( estProbitQuad ) )

# semi-elasticity of age (standard errors of coefficients,
# mean and standard deviation of age provided,
# simplified calculation of standard error)
```

```
urbinEla( coef( estProbitQuad ), xMeanQuad, c( 3, 4 ), model = "probit",
  sqrt( diag( vcov( estProbitQuad ) ) ),
  xMeanSd = c( mean( Mroz87$age ), sd( Mroz87$age ) ) )


######################################################################
####################  urbinElaInt()  #################################
######################################################################
# semi-elasticity of age (full covariance matrix of coefficients)
urbinElaInt( coef( estProbitInt ), xMeanInt,
  c( 3, 4, 0, 5 ), c( 30, 37.5, 44.5, 52.5, 60 ),
  model = "probit", vcov( estProbitInt ) )

# semi-elasticity of age (only standard errors of coefficients)
urbinElaInt( coef( estProbitInt ), xMeanInt,
  c( 3, 4, 0, 5 ), c( 30, 37.5, 44.5, 52.5, 60 ),
  model = "probit", sqrt( diag( vcov( estProbitInt ) ) ) )


######################################################################
####################  urbinEffInt()  #################################
######################################################################
# effect of age changing from the 30-40 interval to the 50-60 interval
# (full covariance matrix of coefficients)
urbinEffInt( coef( estProbitQuad ), xMeanQuad, c( 3, 4 ),
  c( 30, 40 ), c( 50, 60 ), model = "probit",
  vcov( estProbitQuad ) )

# effect of age changing from the 30-40 interval to the 50-60 interval
# (with standard errors of coefficients as well as
# mean and standard deviation of age)
urbinEffInt( coef( estProbitQuad ), xMeanQuad, c( 3, 4 ),
  c( 30, 40 ), c( 50, 60 ), model = "probit",
  sqrt( diag( vcov( estProbitQuad ) ) ),
  xMeanSd = c( mean( Mroz87$age ), sd( Mroz87$age ) ) )


######################################################################
####################  urbinEffCat()  #################################
######################################################################
# effect of age changing from 38-52 years (2nd category + reference category)
# to 53-60 years (3rd category) (with full covariance matrix)
urbinEffCat( coef( estProbitInt ), xMeanInt, c( 3:5 ), c( 0, -1, 1, -1 ),
  model = "probit", vcov( estProbitInt ) )

# effect of age changing from 38-52 years (2nd category + reference category)
# to 53-60 years (3rd category) (with standard errors only)
urbinEffCat( coef( estProbitInt ), xMeanInt, c( 3:5 ), c( 0, -1, 1, -1 ),
  model = "probit", sqrt( diag( vcov( estProbitInt ) ) ) )
```

Index